

# Dynamic Railway Junction Rescheduling using Population Based Ant Colony Optimisation

Jayne Eaton

Centre for Computational Intelligence (CCI)  
School of Computer Science and Informatics  
De Montfort University  
The Gateway, Leicester LE1 9BH, UK  
Email: jayne.eaton@email.dmu.ac.uk

Shengxiang Yang

Centre for Computational Intelligence (CCI)  
School of Computer Science and Informatics  
De Montfort University  
The Gateway, Leicester LE1 9BH, UK  
Email: syang@dmu.ac.uk

**Abstract**—Efficient rescheduling after a perturbation is an important concern of the railway industry. Extreme delays can result in large fines for the train company as well as dissatisfied customers. The problem is exacerbated by the fact that it is a dynamic one; more timetabled trains may be arriving as the perturbed trains are waiting to be rescheduled. The new trains may have different priorities to the existing trains and thus the rescheduling problem is a dynamic one that changes over time. The aim of this research is to apply a population-based ant colony optimisation algorithm to address this dynamic railway junction rescheduling problem using a simulator modelled on a real-world junction in the UK railway network. The results are promising; the algorithm performs well, particularly when the dynamic changes are of a high magnitude and frequency.

## I. INTRODUCTION

Train timetables are designed to ensure the conflict-free running of the railway network; however, in the real world delays may be caused by factors such as train failures, crew shortages, excessive dwell time at the station and obstructions on the line. A late arriving train may miss its scheduled time-slot at a junction or station and will have an increased likelihood of causing conflict with other trains and of propagating its delay throughout the network.

The problem is further complicated by the fact that, while a train controller is trying to minimise delay at a particular point in time, more trains will be arriving at the affected area. These trains may have different priorities to those already waiting to be rescheduled, which makes the problem a dynamic one that changes over time. The rescheduling of trains after a perturbation is usually dealt with by human controllers [6], who often use simple rules such as First Come First Served (FCFS) [4]. Although FCFS may resolve the immediate problem, it may not be the optimal solution in terms of minimising the effect of a train delay in a dynamically changing environment.

The aim of this paper is to investigate the application of Ant Colony Optimisation (ACO) to the problem of rescheduling trains at a junction after a perturbation in a dynamically changing environment. This is referred to as the dynamic railway junction rescheduling problem (DRJRP) in this paper. In the DRJRP, the environmental change is a result of the arrival of new timetabled trains while the original trains are

waiting to be rescheduled at the junction.

In the following sections, we first consider previous work in the area of train scheduling and rescheduling using evolutionary computation (EC) techniques. We follow this with an explanation of the DRJRP followed by a description of the simulator created to model the problem. We then consider ACO algorithms and give details of the ACO algorithm used in this research to solve the DRJRP. Finally, we describe an experimental study carried out to test the ability of the algorithm to solve the DRJRP. The results suggest that ACO is a promising solution to this DRJRP.

## II. RELATED WORK

EC techniques are a group of techniques inspired by nature, which imitate evolution and natural self-organised systems. They include, among others, genetic algorithms (GAs), Ant Colony Optimisation (ACO), evolutionary strategies and particle swarm optimisation.

There has been previous promising work on both train scheduling and train rescheduling using EC techniques. Gorman [7] combined a tabu-search with a GA to produce an optimised schedule for a major US freight railroad with the objective of minimising operating costs. The schedule produced had a potential cost saving of 4% and a reduction in service delay of 6%.

Tormos *et al.* [17] addressed the difficult problem of adding new trains to a train schedule without affecting the existing trains. Their objective was to minimise overall delay. Using a GA they produced a timetable solution in around 300 seconds and their system outperformed comparison algorithms based on random sampling and on Regret Biased Based Random Sampling (RBRS). The GA created has been embedded into a computer-aided tool that is being successfully used by the Spanish Manager of Railway Infrastructure.

Qin *et al.* [16] used a Differential Evolution algorithm to schedule a 185km double-track section of the Shenyang-Siping railway corridor, while Abbas-Turki *et al.* [2] used a GA to tackle the problem of scheduling high speed trains on the Thameslink route in London.

In contrast to train scheduling, which involves creating a fixed timetable of train times without conflict, rescheduling

is concerned with recovering the railway timetable after a disruption. There have been some interesting approaches to solving the problem using EC techniques. Khan *et al.* [11] used a GA to reschedule trains after a delay produced by randomly varying the departure and arrival times for trains on a simulated single track railway section. The GA produced a solution that reduced the train delay at the destination station from 35 minutes to 12 minutes.

A number of researchers have looked at the problem of re-sequencing trains at a junction after a delay. Ho and Yeung [10] encoded a GA to tackle the problem of creating a feasible sequence of train to pass through a junction to minimise conflict after a train delay. They found that their algorithm could produce a solution within less than 5% of the optimal and with a reduced computation time compared to a solution produced using dynamic programming.

Fan *et al.* [6] also considered the problem of sequencing trains through a junction after a delay. They applied both a GA and an ACO algorithm and compared the results to FCFS and a brute force algorithm. Brute force will always find a solution as it involves enumerating all possible solutions. They found both the GA and ACO performed well on the static junction problem although ACO performed slightly better with a smaller computation time

Chen *et al.* [3] used a modified Differential Evolution GA to tackle the problem of rescheduling trains after a delay at the St Pancras Midland Road Junction, which has 3 routes and 2 conflict points. The aim was to reschedule 24 trains in a one hour time-window. They found that their algorithm performed significantly better, in terms of minimising passenger delay, than FCFS on both short and long delay test scenarios.

The above research shows the potential of EC techniques for the scheduling and rescheduling of trains. However, in every case, the problem considered is a static one. In the real world, the rescheduling of trains after a perturbation does not exist in an isolated bubble; while trains are waiting to be rescheduled at the junction more trains could be arriving, and their arrival will change the nature of the problem over time.

### III. THE DYNAMIC RAILWAY JUNCTION RESCHEDULING PROBLEM

#### A. The Description of the Problem

The DRJRP under consideration is based on a static benchmark scenario created by Fan *et al.* [6]. It is concerned with a section of track on the Derby to Birmingham line which takes in the North Stafford and Stenson Junctions. Both the junctions are ‘flat junctions’ in that the merging railroad tracks require that other trains cross over in front of opposing trains on the same level. Two trains can pass through the junction at the same time as long as this does not cause conflict with any other trains. In this paper, the benchmark scenario has been extended to make it a dynamic rescheduling problem by introducing more timetabled trains while the original trains are waiting to be rescheduled at the junction. The perturbation to the system is based on the second of Fan *et al.*’s delay scenarios [6]. Figure 1 shows a diagram of the static junction with the perturbation. In this scenario, the disruption is caused by train 1 being delayed by 5 minutes, which means train 7 arrives before it on track A.

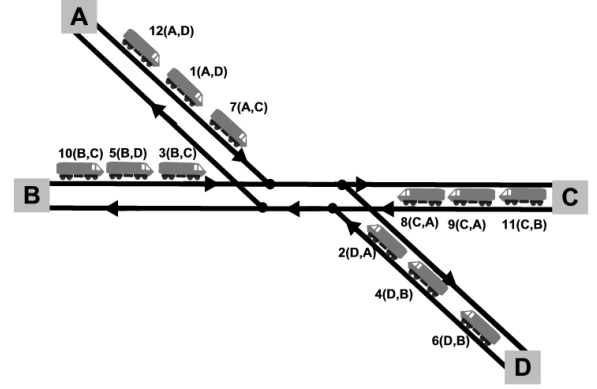


Fig. 1. The junction before a dynamic change.

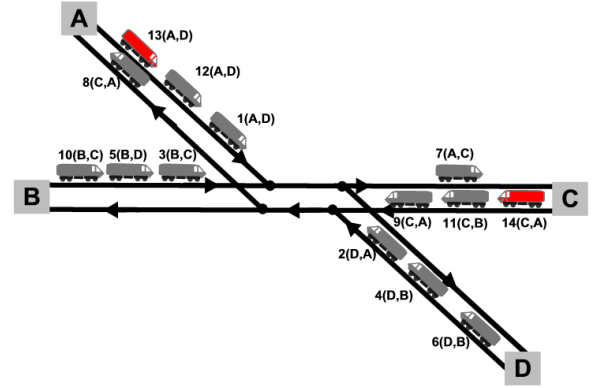


Fig. 2. The junction after a dynamic change.

Figure 2 shows the junction after a dynamic change. Trains 7 and 8 have passed through the junction, but more timetabled trains have arrived while the remainder of the trains are waiting to be rescheduled. Train 13 has arrived on route A while train 14 has arrived on route C. The problem has changed as there is now a different combination of trains to sequence through the junction.

Each train has a delay penalty associated with it, which is the cost in pounds sterling per minute that a train company has to pay if the train is delayed. This is the same objective used by Fan *et al.* [6]. The aim is to find the best order of trains to pass through the junction that minimises the overall cost of the delay.

#### B. The Stenson Junction Train Simulator

Any optimisation algorithm requires a means to test the effectiveness of its solutions in terms of the problem objective. In order to achieve this, a train simulator has been developed, which allows the trains in each train sequencing solution to be run through the simulator to obtain the total delay penalty for that order of trains.

As in the original benchmark scenario used in [6], it is

assumed that the junction is clear at the start of the simulation and that each train begins at set distance from the junction. However, in contrast to Fan *et al.*'s simulator, which used a moving block technology, this simulator uses an automatic fixed block technology to prevent trains from overtaking or running into each other. The moving block technology assumes a safe distance is calculated around each moving train by an area computer which knows the location of all other trains in the area, whereas our automatic block technology works by preventing trains from entering track sections already occupied by other trains. Modelling the junctions using the automatic block technology is an attempt to make the simulation of the junctions as realistic as possible; the moving block technology has limited implementation in the UK railway network although it is in use on a few lines, such as the Docklands Light Railway in London [1].

The resolution of conflict at each junction is modelled by a simulated interlocking system. This prevents a train from entering the junction unless it is safe to do so and is necessary because trains on some routes cross the path of trains on other routes. For example, an examination of Fig. 1 reveals that if train 1 is moving through the junction from A to D it will block all trains on tracks C and B. However, it has no effect on trains travelling from D to A.

Two assumptions have been made when creating the simulator. The first is that trains are not allowed to enter the junction unless both the whole junction and the track section after the junction is clear. This prevents trains from sitting on the track section between the two junctions and causing gridlock. The second is that, to allow the trains to start at the specified distances from the junction, some of the track sections are very short. This means that in some cases the track sections are too short for a train travelling at maximum speed to have enough room to slow down in time if the next track section is blocked. Therefore, maximum speed limits on the short track sections are imposed which give enough time for the train with the smallest braking force to slow down in the distance available. The maximum speed limit through the actual junction is restricted to 64km/h as specified in [6].

Each 'tick', or movement of trains in the simulator, represents one second of time. The speed of the train at each time step is calculated using the Improved Euler Integration, also called Heun's Method. This allows the current acceleration and an estimate of the future acceleration to be combined to find the current speed of the train. Using this method allows for non-constant acceleration. The acceleration of a train at time  $t$  is calculated using Newton's Second Law of Motion ( $F = ma$ ) and the power and resistance tables supplied by Kirkwood and Roberts [12], based on RailSys data. RailSys is used by Network Rail as a simulation tool [19]. Deceleration is a constant maximum brake force for each type of train as in Fan *et al.* [6].

As each train moves along its current track, it checks the status of its next track section. If the next track is occupied, the train will start to slow down when it reaches its stopping distance from the end of the track and will continue to slow down until it stops. If the track ahead is clear, the train will carry on, unless the track ahead has a lower speed limit than the current speed of the train, in which case the train will slow down until it reaches the required speed. If not slowing down

TABLE I. THE SCHEDULED TIMETABLE FOR EACH TRAIN WITH DELAY PENALTIES (BASED ON [6])

Train Number	Train Type	Route	Delay Penalty (£/min)	Scheduled Arrival
1	Class 150	A to D	20	12:10
2	Class 220	D to A	40	12:12
3	Freight	B to C	10	12:19
4	Class 220	D to B	40	12:15
5	Freight	B to D	10	12:20
6	Class 150	D to B	20	12:19
7	Freight	A to C	10	12:28
8	Class 150	C to A	20	12:22
9	Class 220	C to A	40	12:27
10	Class 220	B to C	40	12:32
11	Freight	C to B	10	12:39
12	Class 150	A to D	20	12:36

or speeding up, a train will travel at its maximum speed or the maximum speed of its current track section, whichever is lower.

Table I shows the trains used, their routes through the junction, the penalty for delay and their scheduled arrival times. The delay penalties are different for each type of train and are taken from Fan *et al.* [6]. The timetable was created by running all trains in numerical order through the simulator and recording their arrival times. This gave a base line measurement to be able to calculate the delay of the trains after a perturbation. Each train is one of three types: a Class 150 with a maximum running speed of 120km/h, a Class 200 with a maximum running speed of 200km/h, or a F2-mixed freight train with a maximum running speed of 110km/h [6]. Each type of train is of a different length, the Class 150 is 80.24m long, the Class 220 is 187.4m long and the F2-Freight train is 355m long. The length of a train as well as its speed affects how long the train takes to clear its previous track section. This in turn determines how quickly the following train can move into the train's vacated track.

Dynamism was introduced to the simulator by adding a specified number of trains ( $m$ ) at a specified time interval ( $f$ ). The number of trains added represents the magnitude of change, and the time interval relates to the frequency of change. The new trains are created from the first  $m$  trains in the original timetable. The extra trains can be thought of as an extended timetable for the train junction and each combination of magnitude and frequency is run through the simulator in order to obtain the conflict-free timetable. All new trains are placed at the stations and are not allowed onto the track until the track section leaving the station is clear. At the point of change, any trains that are about to move into, or have moved into, the junction are removed by the simulator from the set of trains that need to be passed to the algorithm.

Figure 3 shows a screen shot of the simulator. The text in the figure has been enlarged to make it easier to see. To be able to observe the movement of trains through the junction, the junction track sections are on a larger scale than the surrounding railway lines. The simulator was constructed using C++ Visual Studio 2012 with a graphical interface created using OpenGL. On a desktop computer with a dual core

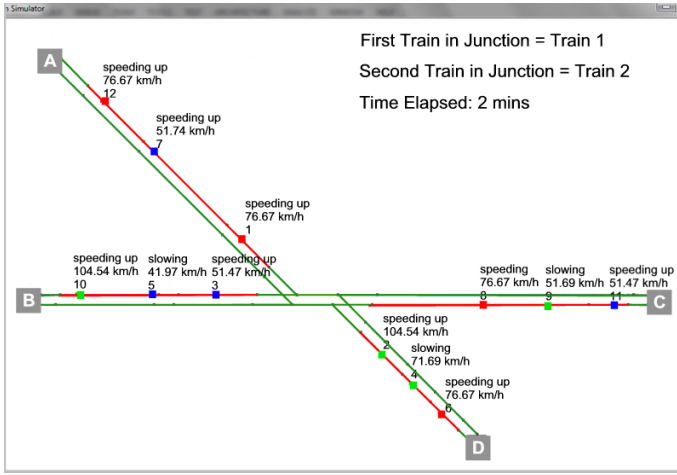


Fig. 3. The junction simulator.

3.2GHz processor and 6GB of RAM, it takes approximately 3 seconds to evaluate one sequencing solution.

#### IV. ACO FOR DYNAMIC OPTIMIZATION PROBLEMS

##### A. Basic ACO Algorithm

ACO is an optimisation algorithm inspired by the ability of ants to follow pheromone trails laid down by other ants to discover food [5]. As ants move backwards and forwards from the nest to a food source they lay down pheromones on the ground which can be sensed by other ants. Ants choosing the shortest path to the food source will return quicker which ensures that the shortest path accumulates more pheromone. Ants tend to probabilistically choose paths with the strongest pheromone concentration which means that a path with high pheromone levels will attract more ants and accumulate even more pheromone. In this way, the shortest path to a food source is marked by the strongest pheromone trail. However, if this trail were to persist after the food source was depleted, it would seriously hamper the ants' ability to find food. Therefore, pheromone trails evaporate over time to allow old decisions to be forgotten.

To apply this principle to an optimisation problem, it has to first be decomposed into a fully connected weighted graph  $G = (V, E)$  where  $V$  is a set of vertexes or nodes, and  $E$  is a set of edges or connections between the nodes. The ants move along the edges of the graph from node to node recording the nodes visited. This list of visited nodes, sometimes called the ant's tour, is one possible solution to the optimisation problem. Pheromones are deposited on the edges of the graph by the ants according to how good an ant's solution is in terms of the optimisation objective. On the next iteration, the updated pheromone levels help to guide the ants to choose better nodes. Pheromones can be decreased as well as increased to model the process of evaporation which allows previous bad decisions to be forgotten. In addition to the pheromone the edges may also be associated with a heuristic value, which is based on problem specific knowledge and provides additional guidance to the ants.

Ants choose the next node probabilistically as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta} \quad \text{if } j \in N_i^k \quad (1)$$

where  $\tau_{ij}$  is the pheromone information and  $\eta_{ij}$  is the heuristic information,  $\alpha$  and  $\beta$  are constants which determine the relative influence of the pheromone and the heuristic values respectively. An ant chooses the next node in this way with a probability of  $1 - q_0$ ; otherwise it chooses the next best node in terms of the pheromone and heuristic values.

##### B. Population-Based ACO (P-ACO)

The above algorithm, however, does not provide any mechanism for allowing the ants to adapt to a change in the environment. Once the ants have converged on a solution, the resulting loss in diversity will make it difficult for them to adapt to a change in the problem and, in addition, the pheromone trails laid down for the previous environment may not provide any useful guidance to the ants in the new environment [14]. One option is to restart the algorithm after a change but such an action is not only computationally wasteful but also results in the loss of information that has the potential to be useful in the new environment.

To address this problem, Guntch and Middendorf [9] introduced a Population based ACO (P-ACO) algorithm. In this algorithm, the best ant found at each iteration is stored in a memory, called the population-list, and only the ants in this list are used to update the pheromone levels. When the population-list reaches its designated limit, an ant is removed and the pheromone trail for that ant is negatively updated. This provides a mechanism for allowing previous bad decisions to be forgotten. To prevent the pheromone levels from building up to a level which means that all ants follow the same path, the amount of pheromone on each edge is bounded between a minimum and a maximum value.

This memory of best iteration ants means that solutions made before the change can be retained to provide valuable information for the new environment. However, to make the ants suitable for the new environment, they may have to undergo a repair operation. Once repaired, the pheromone information for the new environment can be computed from the tours of the fittest ants created before the change, thus ensuring that information from the previous environment can be passed over into the new environment. Guntch and Middendorf [9] found P-ACO to perform better than restarting the algorithm when the environment change was small and frequent and comparable with restart when the change was large and slow.

##### C. ACO for DTSPs

There has been little work on using ACO for dynamic train rescheduling problems. As previously mentioned, Fan *et al.* [6] used ACO for the Stenson Junction benchmark problem with promising results. However, it was a static problem.

There is a similarity between this DRJRP and a Dynamic Travelling Salesman Problem (DTSP). A static Travelling Salesman Problem (TSP) involves finding the shortest route for a salesman to visit a set of cities; it can be made dynamic by changing the number of cities [8], [14], [13], or by changing

the distances between cities [15] over time. In both the DRJRP under investigation and the DTSP, the objective is to find the best sequence of nodes (trains or cities) that minimises an objective.

Several researchers have applied ACO to DTSPs [8], [14], [13], [15]. Here, one issue is that once the ants have converged on a solution they will still follow the same pheromone trails after a dynamic change unless the trails are updated in some way to take into account the new environment. Guntsch and Middendorf [8] tackled this problem by modifying the pheromone trails after a change, either globally or local to the city being removed or added. However, the study involved only the insertion or deletion of one city at a time and they acknowledge that the results may have been different with multiple insertions or deletions. In addition, their solution requires knowledge of where the change has taken place in order to identify the pheromone trails in the local area. Mavrovouniotis and Yang [14] also applied ACO to the DTSP, where the dynamic environment was generated by removing half of the cities from the problem and replacing existing cities with the removed cities. Again the number of cities in the problem does not change overall as the number of cities removed is the same as the number of cities replaced. They found that an ACO algorithm, modified with a local search scheme, performed well on this problem. This previous work on the DTSP suggests that ACO may be applicable to the DRJRP.

## V. PROPOSED P-ACO ALGORITHM FOR THE DRJRP

### A. Framework of the Proposed Algorithm

It is apparent that P-ACO has the potential for solving the DRJRP. In this case, nodes that the ants visit represent trains that need to be rescheduled. The resulting ant tour is the list of trains in the order they are allowed to pass through the junction. The issue is how to represent the problem in a way that makes it possible for the ants to create their solutions while taking into account the fact that if the ants are allowed to visit any node in any order there is nothing to prevent them from creating infeasible solutions where a train is sequenced before the train in front of it.

To resolve this the problem was decomposed into a fully connected, partly one-directional, weighted graph which has the ability to prevent an ant from making an infeasible tour (see Fig. 4). It is based on the design used by Van Der Zwaan and Marques [18] for the job-shop scheduling problem. At any one time, an ant only has the choice of one of the four trains sitting on the four access points into the junction. Each row in the matrix represents one of the tracks that leads to the junction and is one-directional. This prevents an ant from choosing an infeasible train, for example, train 5 before train 3. For ease of reference, this graph will be referred to as a node matrix in the remainder of this paper.

Node 0 represents the start node. At the beginning of each iteration, all ants are placed on this node. They then make a choice about which train node to choose next. After selecting a node, the next train on that train's track becomes visible to the ant and is included in its next decision. After all nodes have been selected, the ant's tour is complete and the ant solution is evaluated by running it through the simulator. The best ant

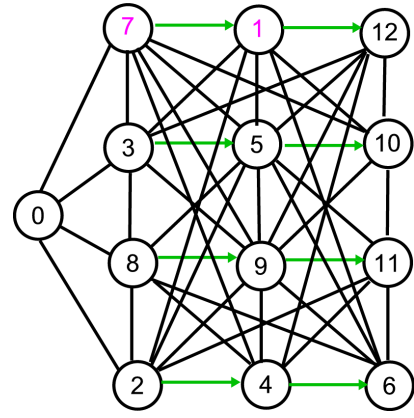


Fig. 4. The initial node matrix.

of the iteration is stored in memory and the pheromone is positively updated for that ant's tour.

In this implementation, the ants rely only on the pheromone values to guide them while making their choices and the value of  $\beta$  is set to zero. A computationally efficient and effective problem-specific heuristic is not available. The natural choice for a heuristic would be the delay caused by sequencing each train. However, the delay of each train is dependent on the sequence of trains that went before it through the junction and is extremely difficult to establish as it changes for each ant's tour. An attempt was made to create an adaptive heuristic that builds the delay values as the ants sequence the trains, but this is extremely computationally expensive and performs worse than using the pheromone alone. The reason for the deterioration in performance is because the ants are unable to distinguish between the two types of delay present in the problem: the delay caused by the perturbation and the delay due to sequencing the trains in a particular order. An advantage of using only the pheromone values to guide the ants is that it reduces the amount of problem-specific knowledge needed to run the algorithm.

### B. Ant Removal Strategy

The fact that the algorithm has a memory raises the question of which ant to remove from the memory once it reaches its pre-set capacity. An experiment carried out to establish whether to remove the oldest ant (Age Strategy) or the worst ant (Quality Strategy) from the memory revealed that removing the worst ant gave slightly better performance over removing the oldest ant. Figure 5 shows the delay penalty for both the Age and Quality strategies averaged over 10 runs of the algorithm.

This is in accordance with the research carried out by Guntsch and Middendorf [9], who found that, in a problem where the ants rely purely on the pheromone to guide them, removing the worst ant solution from the memory results in a good performance; possibly because it provides strong and fairly consistent guidance. This strategy also ensures that the best ant of all the iterations is retained in the memory and provides the elitism that Guntsch and Middendorf [9] found beneficial when running P-ACO without a heuristic. They suggested that the elitism goes some way to providing the guidance that is missing due to the lack of a heuristic.



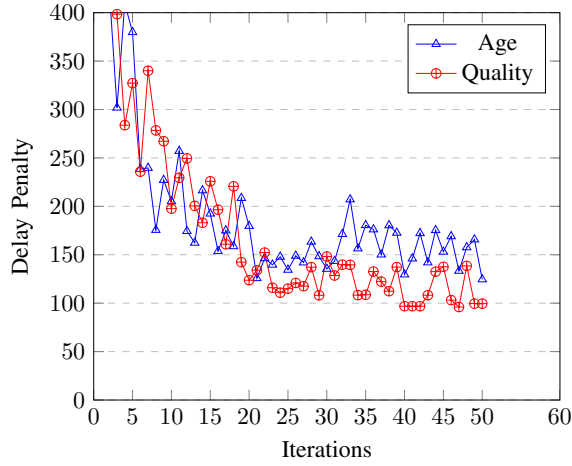


Fig. 5. A comparison of ant removal strategies.

### C. Repairing Memory After a Dynamic Change

To make use of the information held by the ant solutions in memory, the ants' tours have to be repaired. This is achieved by removing any train in an ant's tour that has been removed by the train simulator because it is about to pass into, or has passed into, the junction, and adding the new trains to the end of the solution in the order dictated by the train timetable. This is similar to the KeepElitist strategy used by Guntsch and Middendorf [8] for the DTSP, where cities that are no longer present in the environment are removed and new cities are placed where they cause minimum increase in the distance between cities.

### D. Dynamics Implementation

Even though the trains and junctions are simulated, this is a real-world problem and requires consideration of how it could be implemented in a real-world delayed-train scenario. The supposition is that after a perturbation the algorithm is run very quickly in parallel on several computers in order to find a solution as near optimal as possible in the time available. Ant algorithms are very suitable for running in parallel [5] and doing so would allow a solution to be found in a realistic time.

The sequence of trains in the best solution is then run through the junction until the dynamic change occurs. This change is triggered by the arrival of more timetabled trains. At the point of change a 'snapshot' is taken of the junctions by the simulator. This records the status of the trains, track and junction at that moment in time. The snapshot, plus the new trains, is passed to the P-ACO algorithm, and the algorithm is run again to find the best solution for the new environment. In this way, the algorithm and the simulator are very loosely coupled. The algorithm only acts on the information given to it and does not influence the simulator in any way. This has the advantage that both the simulator and the algorithm can be modified independently of each other.

When the algorithm receives the updated train information, it reconstructs the node matrix to reflect the trains in the simulator snapshot. Any trains that have been removed from the snapshot are also removed from the node matrix and node 0

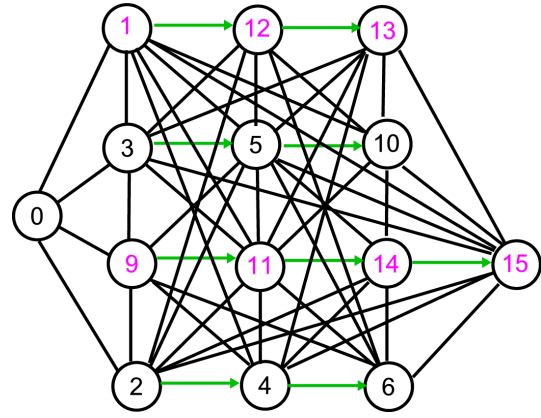


Fig. 6. The node matrix after a dynamic change.

is reconnected to the next four trains sitting at the junction. The new trains are added to the matrix on the row that represents their route into the junction. This means that the number of nodes in the matrix varies dynamically over time.

An example of the node matrix after a change is shown in Fig. 6. Trains 7 and 8 have been removed from the matrix as they have passed through the junction but trains 13, 14 and 15 have been placed in the node matrix on the row that corresponds to their route into the junction.

## VI. EXPERIMENTAL STUDY

An experimental study is carried out to investigate the ability of our proposed P-ACO algorithm to solve the DRJRP.

### A. Experimental Setting

The following pheromone parameters were implemented, as recommended by Guntsch and Middendorf [9]. The maximum pheromone value ( $\tau_{max}$ ) was set to 1, the minimum pheromone value ( $\tau_{init}$ ) was set to  $1/n$ , where  $n$  is the number of nodes, and the pheromone update value to  $(\tau_{max} - \tau_{init})/k$ , where  $k$  is the size of the memory. All pheromone levels were initialised to  $\tau_{init}$ .

The other parameters were established by preliminary experimentation. The best combination was found to be 12 ants with a memory size of 6 and a  $q_0$  value of 0.1. After 150 iterations, very little improvement was found to occur in the ants' solutions. Therefore, the algorithm was run for 150 iterations before each dynamic change.

### B. Performance Measure

In an ideal world, the optimal solution would be available in advance to allow the effectiveness of the algorithm to be evaluated. Establishing the optimum would involve some form of brute force algorithm. There are 369,600 feasible sequences of 12 trains, on four routes, that can be created as possible solutions to the static problem [6]. Each solution takes approximately 3 seconds to evaluate in the simulator which means the static problem alone would take approximately 12.83 days to evaluate all the possible solutions to find the optimal solution. The additional trains added in the dynamic problem would increase the number of possible feasible solutions and would

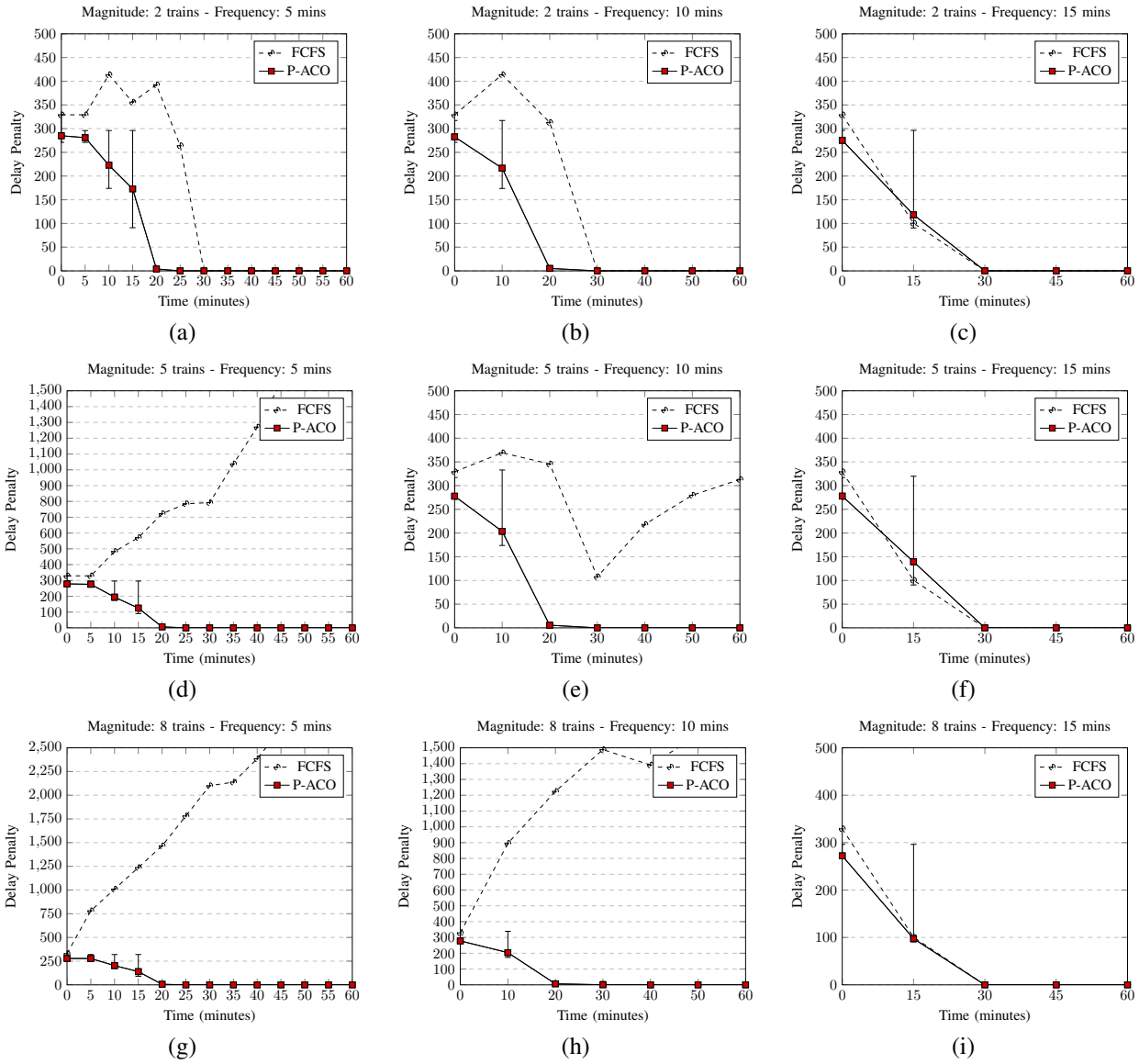


Fig. 7. A comparison between the performance of FCFS and P-ACO on each of the dynamic rescheduling problems for Scenario 2

give an exponential increase in the time needed to find the optimal solutions.

For this reason, the performance of the proposed algorithm is instead evaluated against a FCFS algorithm where trains are assigned to the junction in the order that they arrive at the junction. This performance measure is commonly used by railway controllers to reschedule trains after a perturbation [4] and has previously been used by [6] and [3] to evaluate train rescheduling algorithms.

### C. Experimental Results

Nine different dynamic environments were investigated involving all permutations of 3 different magnitudes of change (2 trains, 5 trains, 8 trains) and 3 different change frequencies (5 mins, 10 mins, 15 mins). For both the P-ACO and FCFS algorithms the total delay penalty at the point of change was recorded. After the last dynamic change, the algorithm was run for a further 150 iterations and the delay penalty was

recorded at the end of the iteration period. The total delay penalty recorded did not include the delay of any trains that had been removed from the consideration by the algorithms because they were about to pass through or had passed through the junction.

Thirty runs were completed for each dynamic environment and the results averaged. Figure 7 shows the outcome for each of the nine combinations of magnitude and frequency. The dashed line represents the delay penalty using FCFS while the unbroken line represents the delay penalty using P-ACO. The vertical lines indicate the maximum and minimum delay penalties produced by the P-ACO algorithm to give an indication of variance. The scale of the different graphs varies to accommodate the maximum delay penalty.

It is apparent from the results that P-ACO outperforms FCFS in all cases where the frequency of change is high, irrespective of the magnitude of change. However, the largest improvement in performance is seen when the dynamic change

is not only of a high frequency but also of a high magnitude. For example, when 8 trains are added every 5 minutes (Fig. 7(g)), the average delay penalty is £277.99 for P-ACO and £781.33 for FCFS after the first change, and is £203.37 for P-ACO and £1009.67 for FCFS after the second change. In addition, after 20 minutes of changes, the effect of the perturbation caused by train 7 arriving before train 1 has been mitigated by P-ACO but persists for FCFS and continues to increase. In the case of low frequency changes (Figs. 7(c), (f) and (i)), the difference between P-ACO and FCFS is minimal with FCFS even showing a small improvement over P-ACO. This may be because of the occasional stagnation observed in the ant solutions while running the experiments. On some runs, the ants were unable to break free of a previously found good solution even though the solution was inferior to solutions found during other runs of the algorithm. This can be a side effect of updating the memory using the Quality Strategy [9]. Addressing this issue may improve the performance of the algorithm even further.

## VII. CONCLUSIONS AND FUTURE WORK

Rescheduling trains after perturbations in dynamic environments is a challenging task. This paper investigates the train rescheduling problem at a railway junction in dynamic environments, i.e., the DRJRP. A simulator is developed to simulate the DRJSP based on a real-world junction in the UK railway network. A population based ACO algorithm is proposed to address this DRJRP. An experimental study was conducted to investigate the performance of the proposed P-ACO algorithm in comparison with a FCFS scheme on a series of DRJRP instances. The experimental results show that the proposed P-ACO algorithm seems to provide a promising solution to the DRJRP, especially when the changes are of high magnitude and high frequency.

The next stage in the investigation will involve research into effective ways to limit the stagnation behaviour occasionally displayed by the ants as a result of removing the worst solutions from memory. A possible way forwards could be to introduce random immigrants to increase the diversity of the ants held in memory.

In addition, more work will be carried out to investigate other delay scenarios. It is plausible that after a larger perturbation involving several trains, P-ACO may be beneficial even when low frequency, low magnitude dynamic changes occur.

Eventually the aim is to introduce a second objective into the DRJRP: that of maximising fuel economy. This will make it not just a dynamic problem but a dynamic multi-objective problem and, as many real world problems are of this type, it would be an important step forward in using ACO to address real-world scheduling problems.

## ACKNOWLEDGMENT

The authors would like to thank the following people at the Centre for Computational Intelligence, De Montfort University: Dr Michalis Mavrovouniotis for his advice on Ant Colony Optimisation and Dr Simon Coupland and Steve Ackland for their help and advice on the physics of train simulation. They would also like to thank Dave Kirkwood at University of Birmingham for sharing his expertise on

train simulations and for supplying the power and resistance tables used in the simulation. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/K001310/1.

## REFERENCES

- [1] "Energy efficiency technologies for railways - moving block," [http://www.railway-energy.org/static/Moving\\_block\\_81.php](http://www.railway-energy.org/static/Moving_block_81.php), UIC-International Union of Railways, accessed: 2014-03-24.
- [2] A. Abbas-Turki, E. Zaremba, O. Grunder, and A. El-moudni, "Perfect homogeneous rail traffic: A quick efficient genetic algorithm for high frequency train timetabling," in *Proc. 14th IEEE Int. Conf. Intelligent Transportation Systems*, 2011, pp. 1495–1500.
- [3] L. Chen, F. Schmid, M. Dasigi, B. Ning, C. Roberts, and T. Tang, "Real-time train rescheduling in junction areas," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 224, no. 6, pp. 547–557, 2010.
- [4] A. D'Ariano, D. Pacciarelli, and M. Pranzo, "A branch and bound algorithm for scheduling trains in a railway network," *European Journal of Operational Research*, vol. 183, no. 2, pp. 643–657, 2007.
- [5] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Scituate, MA, USA: Bradford Company, 2004.
- [6] B. Fan, C. Roberts, and P. Weston, "A comparison of algorithms for minimising delay costs in disturbed railway traffic scenarios," *Journal of Rail Transport Planning & Management*, vol. 2, pp. 23–33, 2012.
- [7] M. F. Gorman, "An application of genetic and tabu searches to the freight railroad operating plan problem," *Annals of Operations Research*, vol. 78, pp. 51–69, 1998.
- [8] M. Guntch and M. Middendorf, "Pheromone modification strategies for ant algorithms applied to dynamic TSP," in *EvoWorkshops 2001: Appl. Evol. Comp.*, Springer, 2001, pp. 213–222.
- [9] M. Guntch and M. Middendorf, "Applying population based ACO to dynamic optimization problems," in *Ant Algorithms*. Springer, 2002, pp. 111–122.
- [10] T. Ho and T. Yeung, "Railway junction conflict resolution by genetic algorithm," *Electronics Letters*, vol. 36, no. 8, pp. 771–772, 2000.
- [11] M. Khan, D. Zhang, M. Shi Jun, and Z. J. Li, "An intelligent search technique to train scheduling problem based on genetic algorithm," in *Proc. 2006 Int. Conf. Emerging Technologies (ICET'06)*, 2006, pp. 593–598.
- [12] D. Kirkwood and C. Roberts, "Validation of railway network simulation software," Copenhagen: IAROR 5th Int. Conf. on Railway Operations Modelling and Analysis, 2013.
- [13] M. Mavrovouniotis and S. Yang, "Ant colony optimization with immigrants schemes in dynamic environments," in *Parallel Problem Solving from Nature, PPSN XI*. Springer, 2010, pp. 371–380.
- [14] M. Mavrovouniotis and S. Yang, "A memetic ant colony optimization algorithm for the dynamic travelling salesman problem," *Soft Computing*, vol. 15, no. 7, pp. 1405–1425, 2011.
- [15] M. Mavrovouniotis and S. Yang, "Ant colony optimization algorithms with immigrants schemes for the dynamic travelling salesman problem," in *Evolutionary Computation for Dynamic Optimization Problems*, ser. Studies in Computational Intelligence, S. Yang and X. Yao, Eds., Springer Berlin Heidelberg, 2013, vol. 490, pp. 317–341.
- [16] S. Qin, L.-S. Zhou, and Y.-X. Yue, "A clonal selection based differential evolution algorithm for double-track railway train schedule optimization," in *Proc. 2nd Int. Conf. Advanced Computer Control (ICACC)*, vol. 1, 2010, pp. 155–158.
- [17] P. Tormos, A. Lova, F. Barber, L. Ingolotti, M. Abril, and M. A. Salido, "A genetic algorithm for railway scheduling problems," in *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*. Springer, 2008, pp. 255–276.
- [18] S. Van Der Zwaan and C. Marques, "Ant colony optimisation for job shop scheduling," in *Proc. 3rd Workshop on Genetic Algorithms and Artificial Life (GAAL 99)*, 1999.
- [19] R. Watson, "Train planning in a fragmented railway—a British perspective," Ph.D. Dissertation, © Robert Watson, 2008.